Model-Based Query Systems for Emergency Response

Model-Based Query Systems for Emergency Response

Stephen Potter
AIAI, School of Informatics,
University of Edinburgh, UK
s.potter@ed.ac.uk

Gerhard Wickler
AIAI, School of Informatics,
University of Edinburgh, UK
g.wickler@ed.ac.uk

ABSTRACT

In this paper we describe the approach adopted and experiences gained during a project to develop a general architecture that aims to harness advanced sensor, modelling and Grid technologies to assist emergency responders in tackling emergencies (specifically fire emergencies). Here we focus on the command and control aspects of this architecture, and in particular, on a query-based approach that has been adopted to allow end users to interact with available models of physical and other phenomena. The development of this has provided a number of insights about the use of such models, which along with the approach itself, should be of interest to any considering similar applications.

Keywords

Command-and-Control, Artificial Intelligence, Modelling, Information Systems.

INTRODUCTION

The FireGrid project (Berry *et al.*, 2005) represents a farsighted attempt to harness recent advances in a number of disparate fields for the express purpose of assisting responders to tackle emergency incidents, in particular (but not exclusively so), complex building fires. At present, firefighters, when they arrive on the site of an incident, generally have to rely on the information provided by their own senses, any information that can be provided by evacuated occupants of the building in question and their experiences of previous fires in order to decide on an intervention course. In the UK, the initial decision is one of choosing the appropriate tactical mode for tackling the fire: this may be either *offensive* or *defensive* (HM Fire Service Inspectorate, 2002). The former often involves sending firefighters into the building, a decision that is taken if the potential benefits are felt to outweigh the risks – for example, if people are thought to be trapped within the building and firefighters are felt to have a reasonable chance of rescuing them at an acceptable level of risk to the firefighters themselves. Defensive mode, on the other hand, is adopted when the tradeoff of the potential benefit against the likely risk of offensive mode is not thought favourable (or, in some situations, where the current lack of information means that the benefits or risks cannot yet be assessed).

Hence, the intervention decision can be based on incomplete or faulty information; in particular, for large-scale and complex buildings, firefighters are rarely aware of the exact conditions within the building. Moreover, the lack of experience of complex fires that many firefighters have (simply because such fires occur relatively rarely), can mean that, even when available, information is misinterpreted, and firefighters are placed in danger. Obviously, this is an unsatisfactory state of affairs. However, recent advances in three areas of technology, when exploited together, suggest a solution to this problem:

- Developments in sensor technology, along with a reduction in unit cost, offer the prospect of deploying large-scale, robust and cost-effective sensor networks within buildings;
- Advances in the understanding of fire and related phenomena have resulted in sophisticated computer models which might be used to interpret sensor data;
- The availability of Grid resources and infrastructure promises to enable these (usually extremely time- and resource-hungry) models to be run in real-time, making their use in emergencies a practical proposition.

With the addition of a command and control ‘layer’ to allow responders appropriate access to these technologies, this combined approach suggests a new way of responding to emergencies: this new way of working is the FireGrid vision.

In this paper we choose to concentrate on the command and control (C2) aspects of this architecture, and specifically, the query-based approach that we have adopted to allow the emergency responder to interact with the system to invoke interpretative models, and the implications that this approach has for the use of these models. These models have, in general, been developed in an academic context and have not necessarily been intended for use in emergency situations, which, as will be seen, can impede or prevent entirely their use. We believe that the approach adopted and, perhaps more importantly, the experiences that we have gained to date should be relevant to any in the field of emergency response support who are attempting to apply similar models or, more generally, to exploit 'academic' knowledge of physical and other phenomena for response use.

In the next section we describe the C2 components of a FireGrid system. Central to this is the Query Manager agent that has the task of invoking models to answer requests for information. The use of models in this way is discussed in the following section, which is followed by a discussion of the Query Manager and the language used to specify queries.

THE C2 LAYER

The role of the C2 layer of a FireGrid system is, in brief, to provide a means for users to interact with the system and steer it towards achieving their goal – which, in a deployed system, would be to help with the safe and successful management of fire incidents in the building in question. The unique aspect of a FireGrid system is the capture of 'live' sensor data from the building and the use of this data by models to interpret the status and projected course of the incident for emergency responders. Figure 1 shows the components of the C2 layer.

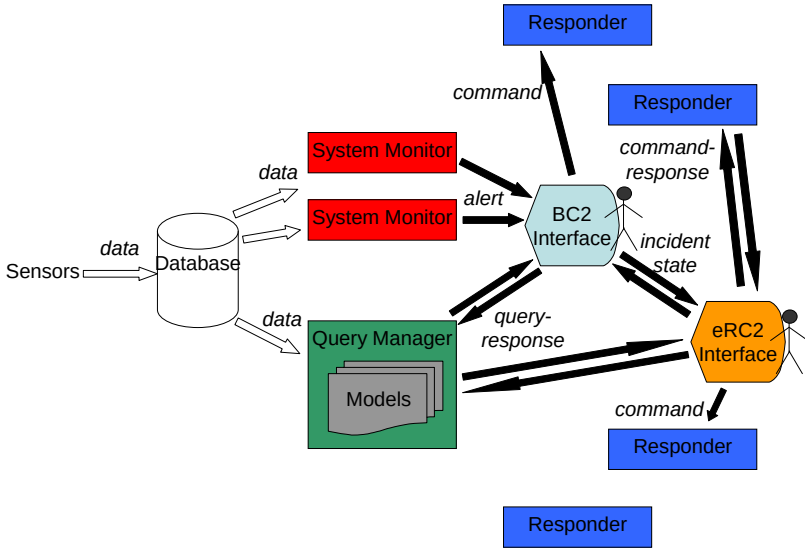

Figure 1. The FireGrid system architecture from a C2 perspective. Arrows show principal communication flows only, with inter-agent communications indicated by solid arrows.

There are two primary human interfaces onto the C2 layer, namely the *Building C2* (BC2) interface, and the *e-Response C2* (eRC2) interface. The role of these interfaces is to provide their human users with information about the current state of the system (and hence about the state of any incident and of the response to it), and to assist users to interact with system components to acquire additional information or actuate some response.

The two types of C2 interfaces differ in their applicability, coverage and scope. A BC2 interface is specific to a particular FireGrid system, and is tailored towards that system and the building it relates to. Its projected user is someone who has responsibility for monitoring the state of the building in question and, in the event of an incident, for instigating initial response activities (such evacuating the building), but would not be expected to tackle anything but the most trivial of fires.

The eRC2 interface, on the other hand, contains knowledge of agents (such as fire-fighters) and resources (such as standard operating procedures) that are external to any specific FireGrid system, and which may be required when

the response to incident has to be escalated beyond the local (that is, BC2) level. The eRC2 interface is intended to be installed on, for instance, the computer system in an emergency response command vehicle; when the vehicle arrives at the site of the incident, it 'taps into' the *in situ* FireGrid system to access and request information about the incident. The projected user of the eRC2 interface is (using UK terminology) a Fire Incident Commander, or – more likely – the Support Officer detailed to assist the Incident Commander. The Incident Commander is responsible for the management of the incident, including tactical planning, coordination and resource deployment (HM Fire Services Inspectorate, 2002).

The system database, which is dedicated to this particular system, is used to store data from the various sensors deployed around the building. In addition this database contains data which is necessary to *interpret* the sensor data in ways useful to the end users; this data describes the physical layout of the building, positions of the sensors and so on. Note, however, that neither the BC2 nor the eRC2 interface accesses this data directly; instead, it is mediated through one of two types of computer agent: the *Query Manager* agent and the *System Monitor* agent(s).

The Query Manager is an autonomous agent, specific to a particular FireGrid system; it is expected that every FireGrid system will contain exactly one such agent. It accepts requests for specific information from the user of a BC2 or eRC2 interface, and, where there is available a model able to generate the requested information, it invokes this model, returning its results as the answers to the queries. Hence, users do not interact directly with the data-interpreting models in the system, but instead all their requests for information are directed to and handled by the Query Manager. The Query Manager, and the queries it manages, are the central topics of the following sections.

A System Monitor agent periodically applies some interpretive model to the latest sensor data in order to check for certain potentially hazardous conditions, and when such conditions are found to hold, generates and sends an alert to the C2 interfaces. A simple example might be a 'fire alarm' agent, which examines thermocouple readings for signs of fire. A system may contain any number of such agents. The final type of agent in the C2 view of a FireGrid system are the *Responder* agents; these are agents that, in general, are capable of performing physical intervention activities, of providing information and responding to commands issued by the user of the BC2 or eRC2.

This general structure has been demonstrated with several small-scale applications, involving data collected from real (controlled) fires. The agents in the system have been constructed using the *I-X* approach for emergency response systems (Potter, Tate and Wickler, 2006; Wickler, Tate and Potter, 2006). I-X provides a generic systems architecture (and tool suite) for multi-agent process support, structured upon an abstract activity-centred ontology for expressing information and communications within the system. While this approach has its foundations in work in AI planning, it is intended for use in systems of collaborating human and computer agents.

A key element of a FireGrid system, then, is the Query Manager. This agent fulfils a vital role in the architecture: this is the agent that 'knows' about the existence and availability of information-providing models, knows the types of queries that these models can answer, and knows how to invoke them. It provides the central point of any FireGrid system, and is assumed to always be present within a system. Before discussing the operation of the Query Manager, however, more needs to be said on the use of models for emergency response.

THE USE OF MODELS

In its general aim of increasing situation awareness for responders, FireGrid is similar to a number of recent and current projects, such as DEFACTO (Schurr *et al.*, 2005) and ALADDIN (Rogers *et al.*, 2008), additionally sharing with the latter project the notion of using sensor data to convey the state of the emergency environment. However, FireGrid is unique in its ambition to incorporate models into the response system and allowing these to be steered by sensor data to answer queries about the emergency.

By the term "model" we mean some generalised conceptualisation of aspects of the physical environment that, based on values of certain input parameters, can provide *useful* results; that is, results that are both acceptably correct and relevant to the emergency response. Hence, we are not concerned with *how* the model produces its results but rather in characterising more abstractly what the model does in this context. In general terms, we are thinking of two different types of model: those which interpret (some aspect of) the current state of the incident, and those which predict the future course of (some aspect of) the incident. An example of the former would be a model that interprets the current readings of thermocouples in a room containing a fire to determine the maximum temperature; an example of the latter would be a model that interprets the readings to predict the time at which the ceiling of the room will collapse; and as can be seen from these examples, our notion of model embraces both simple and complex interpretations of the sensor data, and models will invariably be customised to some degree to the particular building

(and its sensor network) in question. The range of models available to a particular deployment of a FireGrid system effectively delimits the capabilities of that system for assisting responders to make decisions.

In practice, these models will often have been developed in an academic context and for purposes other than emergency response, and the use of such models for emergency response raises a number of issues. Such models can rarely be used 'directly', and not all models are appropriate for use in this context. The difference in objectives between academic modelling and emergency response means that, to be incorporated within a FireGrid system, models will generally need to be tailored to the purposes of response and to the context of the particular system in question. For some models, this will involve, to a lesser or greater degree some coding effort; however, other models may simply be not applicable to emergency response at all. This use of models raises a number of issues, discussed briefly below.

Usefulness of Results

The results that the model generates should represent potentially useful information for assisting with the overall aim of the system (that is, the response to an emergency). While in some cases the potential usefulness of a model's results can seem self-evident, in other cases this could be a more difficult quality to judge, and could depend on circumstances and the environment in which the system is located.

Presentation of Results

The importance of the appropriate presentation of results should not be overlooked; poor presentation can render the most useful results useless (or worse – dangerous). The "appropriateness" of a presentation depends on both the fire modelling principles underlying the model – the presentation should not misrepresent what the results actually *mean* – and the end user and the context in which results would be used. Indeed, there may be more than one user of the results, each having different tasks and thus requiring different presentations. The criticality of some results and the pressures of operating in an emergency situation can also have a bearing on the manner in which information is presented. Interfaces can be difficult to design and time-consuming to develop and test.

Model Applicability

The model should require, as input, only that information that can be reasonably expected to be known at during the incident (and at a time when its results would still be useful). In other words, for use in response, models should be able to provide useful information without the need for inputs that are essentially unknowable (or, at least, very uncertain) at the time of the incident. So, for instance, a model that requires input about the physical layout of the building in question would (probably) be applicable, whereas a model that demands precise information about, say, the material on fire, the nature and position of furniture in a room, and so on, would probably not be acceptable, since such information is very unlikely to be available. The lack of this sense of 'generality' in fire models seems to be one of the main areas where the gap between academic research and practical application is felt. For the former, the approach is often (but not always) one of *a posteriori* modelling – based on data gathered during fire experiments, and with knowledge of the precise experimental conditions including the location and fuel of the fire, the modeller will attempt to produce a parameterised model that most closely reproduces the measured fire data. However any such model that is overly sensitive to the precise conditions will not be suited to use for response, where these conditions vary and will not be known to the system.

Another related difficulty that can arise is that the model may not have been designed to work with 'live' data; that is, it assumes the availability of the complete data set, and, moreover, one which has been pre-processed (usually by hand) to rectify noisy and missing data points. The production of models that operate with live data and produce useful results with possibly incomplete and noisy data sets may require different modelling approaches.

Practicality

The particular implementation of the model should be such as to allow results to be generated and communicated in good time. This is difficult to quantify, but means that interpretations and inferences about current state of the incident should arrive while these are still valid (or, at least, useful); and predictions about future states should arrive while the predictions still hold (and certainly before the time referred to by the prediction!). In certain cases the use of Grid/High-Performance Computing resources can serve to make running a particular model a practical proposition (and this thought was one of the initial motivations for the FireGrid project); however, unless the

implementation has been developed with this use in mind, porting the model code to these resources will often require effort.

Re-purposing Effort

Some re-purposing effort will usually be required to adapt a model so that it produces useful results within the constraints of an emergency response context. Models will generally need to be 'wrapped' with other code that, in response to a query for information, generates the expected input (calling to the database for sensor readings and other data) and parses the produced output into an appropriate answer. It may be the case that some of the required input data does not currently reside in the database. If so, and if it is feasible to acquire and add the data in the context of the system, the database schema must then be altered to include the new data (bearing in mind the possibility of unintended side-effects resulting from changes to database design). Furthermore, some characterisation of the wrapped model in terms of the sort of queries it can answer (its competence or *capability*) needs to be expressed to the system so that it can be matched to relevant queries.

Summary

In summary, the development of suitable models requires a very different modelling philosophy, one that leads not merely to accurate models, but to models that are robust and useful (from the perspective of the responder). It seems very unlikely that any model that has not been developed under this philosophy can be usefully adopted by emergency response systems, or at least, not without a significant amount of re-purposing effort.

THE QUERY MANAGER

When it receives a query from a BC2 or eRC2 user, the Query Manager proceeds as follows:

1. The list of available models is searched for those that have the capability of answering the query; if none is found to have this capability, the Query Manager replies with a message to this effect and awaits the next query. We shall have more to say about the description of the capability of a model later.

2. Assuming that one or more capable models are found, one of these is selected, and invoked to produce an answer to the query (the possible criteria for choosing between valid alternatives will also be discussed later).

3. The invoked model returns its answer to the Query Manager, which then passes on the answer to the querying agent. The Query Manager then awaits the arrival of the next query.

A *query*, then, is a request for information at some time about fire incidents occurring in the environment in which a FireGrid system is situated. It should be obvious from the above description that the form and content of the queries is a crucial element of the Query Manager. The 'language' used for constructing queries needs to reconcile a number of often conflicting and complex considerations:

- The language should allow useful (from the perspective of the system user) queries to be expressed, that is, they should represent requests for information that will help make intervention decisions when tackling a fire emergency.
- The language should exclude queries that are not realistic in respect of the data that might feasibly be available during an incident (data from both sensors and facts known *a priori* about the system and its environment) and of the state of the art of modelling.
- The language should allow only queries that are unambiguous and which provide sufficient information to select and invoke the appropriate model.

The following sections describe the query language that has been developed with assistance of both modellers and emergency responders. Before discussing the form of queries themselves, we first introduce some of the concepts that underpin them.

Time and Space

In order to bound the scope of queries so they become tractable, for every query it is necessary to specify both the time (frame) and the physical location to which it refers. In other words, we need to bind requests for information to particular times and places.

To specify time explicitly in queries, we adopt a relative measure (e.g. "120 seconds from now"), rather than asking the querying agent to select a specific absolute time (e.g. "13:26:53 on the 19th December 2007"), since this seems a more natural approach from the perspective of the end user of the system. Note, however, that answers should refer to *absolute* times, since it is important to know exactly when a particular piece of information refers to (and, indeed, when it was given).

A particular FireGrid system will be unique to a specific place (usually a building), and as a result, references to locations in queries can only be made by using application-specific terms. The spatial environment of a system is considered to consist of one or more physical *locations*. A location is defined to be a specific 3-dimensional space. One location may include, wholly or partially, other locations. Every query represents a request for information about the physical conditions *within* a specified location. Ideally, a location should be both a space to which fire models can be applied and a space that is of relevance to emergency responders. Again, these requirements do not necessarily converge; whereas some locations (such as compartments/rooms) may meet both criteria, others (such as escape routes) may require more complex reasoning and/or the invocation of a number of different models.

For a particular application, then, a set of labels – which must be used consistently by all elements of the system – will be introduced for unambiguously designating specific locations (e.g., *kitchen, lounge*). Moreover, a set of physical relationships will be required for relating the locations denoted by each term into the wider space of the deployment (e.g., *kitchen is-next-to lounge, floor-7 contains room-712*) to allow some reasoning about space.

States and Events

Every query is a request for information about either the value of a *state* or the occurrence of an *event*:

- An incident state (variable) is some measurable quantity at some location that persists for some duration of time during the incident, and which is measurable throughout that duration. So, for instance, the height of the smoke layer (keyword *smoke-layer-height-value*) of some location is a state that is continuously measurable during a fire. Associated with every state parameter is a convention for expressing values to which every communication is expected to conform; smoke layer height, for instance, is conventionally measured in metres from the floor of the location in question.
- An event, as defined here, is some phenomenon that is assumed to occur instantaneously (if it at all) at some location during the incident; in other words it has no temporal extension. Examples of event are *flashover, collapse, ignition, explosion*.

Query Types

In the current definition of the language queries can be broadly grouped into two types: those that request information about the *current* status of the incident (*confirm* queries); and those that request information about the projected future states or events (*predict-value, predict-when* and *predict-where* queries) which will be answered by recourse to some predictive model of the evolution of the incident

Confirm queries

A *confirm query* is a request for the current value of one of the state variables that describes the incident; as such, the answer will (probably) be derived from the latest sensor readings. A confirm query has two parameters:

- The name of the state of which the current value is requested. Note that, as it is defined above, it is nonsensical to request the value of an event; since an event is instantaneous, it cannot be said to occur "now", or, for that matter, at any other specific point in time (it can, however, be said to occur within some time interval, as we shall see below).
- The label name of the location to which the query refers.

An example confirm query is:

```
confirm-query smoke-layer-height-value compartment
```

which should be read as "confirm the current value of the smoke-layer-height in the compartment location". The expected answer to this query will be a value of the state parameter in question, given in terms which conform to the agreed convention for that parameter.

Predict-value query

A *predict-value query* corresponds to a request for a prediction of the value of some specified state of the incident at a given location and at a given time. Hence, a query of this sort has three parameters:

- The name of the state parameter for which a prediction is requested. (Analogously to the case for confirm queries, it does not make sense to predict the occurrence of an event at a particular time, since events are assumed to occur instantaneously.)
- The label of the location for which the prediction is to be made.
- The time about which the prediction is to be made. This is specified as a relative value in seconds from the time the query is made. In practice, the interpretation of "now" is somewhat problematic, since the time taken to process the query and produce an answer will not always be negligible – time is taken in communicating the query, requesting the necessary data from the database, running the model and so on – and, of necessity, a somewhat pragmatic attitude to this question has been taken.

An example predict-value query is as follows:

```
predict-value-query smoke-layer-height-value compartment 60
```

to be read as "predict the value of the smoke-layer-height in the location compartment in 60 seconds from now". The expected answer will contain the predicted value of the state parameter, expressed in the appropriate conventional terms.

Predict-when and Predict-where queries

A *predict-when query* is a request to the system to predict the (first) time that some state assumes a particular value or that some event occurs at a given location and within some given timeframe. Hence, this query also has three parameters:

- Either a *state-expression* or an *event-expression*. A state-expression consists of a state parameter, an relational operator and a corresponding value; for example, *smoke-layer-height-value < 1.5*. An event-expression consists of the name of some event (e.g., *explosion*); it is implied that the query is for a prediction of the time of its occurrence.
- The label of the location in question.
- The timeframe of interest in seconds. It is necessary to specify some upper bound on the time for which a model should look into the future when trying to make a prediction, since it will not necessarily be the case that the state will ever assume the specified value or that the event will ever occur, and without a 'hard' cut-off time, a model process might then never terminate, looking further and further into the future.

An example:

```
predict-when-query smoke-layer-height-value<1.5 compartment 600
```

or "predict the first time when the value of the smoke-layer-height is less than 1.5 metres from the floor of the location compartment within a timeframe of the next 600 seconds".

A *predict-where* query is the spatial counterpart of the predict-when query. It is constructed in a similar manner, but the location parameter is assumed to refer to some location that contains sub-locations: it is a request for a prediction of the first of these locations where a state-expression becomes true or event occurs within the given time-frame.

Some Comments on the Query Language

The query language described above represents a pragmatic first attempt to provide a *lingua franca* expressing both the needs of responders and the possibilities of (fire) modelling. However, a number of observations can be made about this language, which due to space constraints, are merely mentioned here:

- There is, as yet, no fully formalised manner of *answering* queries – what is needed is a complementary *response language*, which should be readily incorporated into the wider knowledge representation scheme for the system.
- As they are defined, queries represent requests for 'point facts' – "give me the value of X at location Y and time Z", i.e., the value at coordinate (X, Y, Z). Such a view is quite restrictive, and, in the case of predictions,

ignores possibility vital state changes or events between now and time Z. A richer query language might allow requests for more complex information that represent lines or planes within this 'information space'.

- The query language does not currently provide a means of querying what has already happened – it only allows queries about the current status or predicted future status of the incident.
- Currently queries contain only a description of the required information; it may be necessary to include additional constraints about that information (e.g., "requires a response within 60 seconds").

ON THE USE OF QUERIES

Our approach to the use of models for emergency response has been to situate these within a query-answering framework: end-users pose queries which the Query Manager attempts to answer with the use of models. In this context, we can identify two distinct uses of the query language: the first, and obvious, is its use by the end-user to specify requests for information; the second use is in the description of the query-answering *capability* of a model.

Queries and the End-User

Figure 2 shows a simple interface developed to allow users to construct and submit queries in demonstration deployments of FireGrid systems. However, it is not necessarily intended that the user him- or herself has to manually 'write' every query – queries could, for instance, be formulated in advance for the system in question. It should be possible to construct (with the input of responders) 'canned' queries that both represent requests for typical information that is useful for responders and which can be answered by models available to the system in question. These queries could be stored at the Query Manager and 'run' automatically on notification in response to predefined events (such as the 'arrival' of the eRC2 interface).

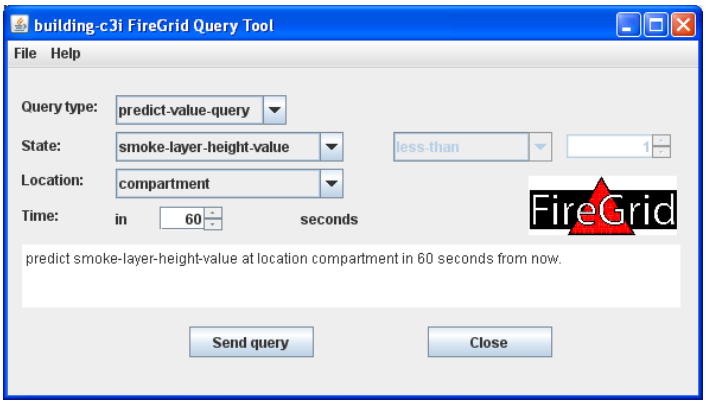

Figure 2. An interface for manually constructing and submitting queries.

Of course, we are likely to always want to provide responders with tools that allow them to generate specific queries pertaining to the state of the incident in question and the intervention courses being considered. Given the pressures of operations during an emergency, the most appropriate interface for generating queries in this manner remains an open question – the 'text-heavy' interface of Figure 2 would seem too cumbersome and would require, among other overheads, its user to know how the location labels in the interface refer to the building before him/her. This is an area where more effort is required.

Queries and Model Capabilities

Vital to the successful use of models is the *capability* of each model – this is a succinct description of the functionality of the model, expressed with respect to the aims of the FireGrid system, that allows the Query Manager to determine whether the model is relevant for answering the current query. A second use of the query language is that it provides a set of terms which allow us to express capabilities. For instance, the capability of a "smoke-layer height prediction model" can, with the addition of a few simple logical terms, be expressed as the rule:

```
IF current_query_type = predict-value-query
       AND current_query_state = smoke-layer-height-value
       AND current_query_location = compartment
THEN TRUE
ELSE FALSE
```

Each rule is applied by the Query Manager to the component terms of the current query, and if the condition (the *IF…* part) holds, the rule returns *TRUE*, that is, the model in question is applicable to this query. If the condition does not hold the rule returns *FALSE*, and the Query Manager discounts the model in this case. Hence, the rule uses the terms of the query language to define the space of possible queries to which this model relates.

In addition to its capability, there are a number of other characteristics of a model that might usefully form part of the description given to the Query Manager, where it could be used to help determine which model to invoke in the event of more than one model having an applicable capability. These characteristics include the availability of the model (the computer on which the model runs may be down, for whatever reason, or is unreachable due to network problems), certainty/accuracy and precision of model results, and model run-time, some indication of the length of time that the model takes to produce a result.

CONCLUSION

This document describes the query-answering approach that has been developed to allow access to computational models to assist in the response to (fire) emergencies. Requests for information are sent in the form of queries to a centralized Query Manager agent that is able to assess whether any available models can provide this information. The formulation of queries, and the descriptions of the capabilities of models, draws upon the concepts and terms of a query language developed expressly for the purposes of this system. With the appropriate modifications, this approach is intended to be applicable to types of emergency other than fire incidents.

The use of models in this system is an appealing one (especially when integrated with live sensor data): the models would seem to embody specific expertise that is placed at the service of the responders. However, in the course of this work, consideration of the nature of models (and of modelling) has revealed the frequent disparity between the models that are available, and the needs of and constraints imposed by emergency response. The issues that this raises will be relevant to any who are attempting to develop similar applications.

ACKNOWLEDGMENTS

The work reported here has been done as part of the FireGrid (www.firegrid.org) project. This project is co-funded by the United Kingdom's Technology Strategy Board's Collaborative Research and Development programme, following an open competition. The authors would like to thank the anonymous ISCRAM reviewers for their thoughtful criticisms of this paper.

REFERENCES

1. Berry, D., Usmani, A., Torero, J., Tate, A., McLaughlin, S., Potter, S., Trew, A., Baxter, R., Bull, M. and Atkinson, M. (2005) FireGrid: Integrated Emergency Response and Fire Safety Engineering for the Future Built Environment, UK e-Science Programme All Hands Meeting (AHM-2005), Nottingham, UK, Sept. 19-22, 2005.
2. HM Fire Service Inspectorate (2002) Fire Service Manual, Volume 2 Fire Service Operations, Incident Command, HM Fire Services Inspectorate Publications, London: The Stationary Office. Crown Copyright.
3. Potter, S., Tate, A. and Wickler, G. (2006) Using I-X Process Panels as Intelligent To-Do Lists for Agent Coordination in Emergency Response, Proceedings of the Information Systems for Crisis Response and Management 2006 (ISCRAM2006), Special Session on Multiagent Systems for Disaster Management and Response, Newark, New Jersey, USA, May 15-17, 2006.
4. Rogers, A., Ramchurn, S.D., Jennings, N.R., Osborne, M.A. and Roberts, S.J. (2008) Information Agents for Pervasive Sensor Networks. To appear in *Proc. 4th IEEE Int. Workshop on Sensor Networks and Systems for Pervasive Computing*, Hong Kong, China, March 2008.
5. Schurr, N., Marecki, J., Lewis, J.P., Tambe, M. and Scerri, P. (2005) The DEFACTO System: Coordinating Human-Agent Teams for the Future of Disaster Response. In *Multi-Agent Programming*, 15, Bordini, R.H., Dastani, M., Dix, J., El Fallah Seghrouchni, A. (Eds.), Springer, 2005.
6. Wickler, G., Tate, A. and Potter, S. (2006) Using the <I-N-C-A> Constraint Model as a Shared Representation of Intentions for Emergency Response, Proceedings of the First International Workshop on Agent Technology for Disaster Management (ATDM), at the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), Future University, Hakodate, Japan, May 8-12, 2006.